A Framework for Image Processing with the DSK6416*

Jacob Fainguelernt

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

This module describes general procedures for reading and writing stills images with the DSK6416. The scripts and models created her will serve as the framework for image processing algorithms simulation and implementation.

1 Introduction

This module describes general procedures for reading and writing stills images. The scripts and models created in this chapter will be the basis for image processing algorithms simulation and implementation using the DSK6416.

1.1 Related Files

- Simulink Model for Simulation - stills_sim.mdl[1]
- Input Image - TMW2.jpg[2]
- MATLAB script for Real-Time - Picture_script.m[3]
- Simulink Model for Real-Time - stills_R_W.mdl[4]

2 Activating CCS under Simulink®

This section will show you how to activate CCS under the Simulink.

1. Create a new Simulink model.
2. Open the Simulink library browser and add the "C6416DSK" from the "Target Preferences" group of the "Embedded Target for the TI TMS320C6000 DSP" Blockset (Please refer to Figure 1).

*Version 1.1: May 26, 2009 11:22 am -0500
†http://creativecommons.org/licenses/by/2.0/
[1]See the file at <http://cnx.org/content/m23996/latest/stills_sim.mdl>
[2]See the file at <http://cnx.org/content/m23996/latest/TMW2.jpg>
[3]See the file at <http://cnx.org/content/m23996/latest/Picture_script.m>
[4]See the file at <http://cnx.org/content/m23996/latest/stills_R_W.mdl>

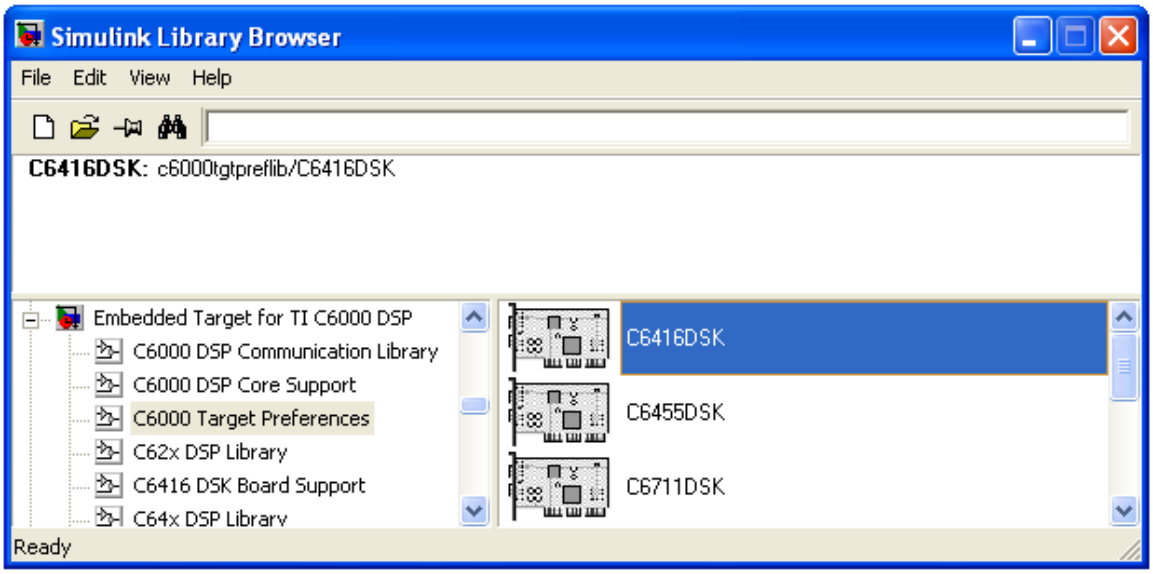


Figure 1: The C6416DSK Block

3. Set the model to "Generate a CCS Project" in the Simulation->Configuration Parameters -> Build Action, as shown in the following picture:

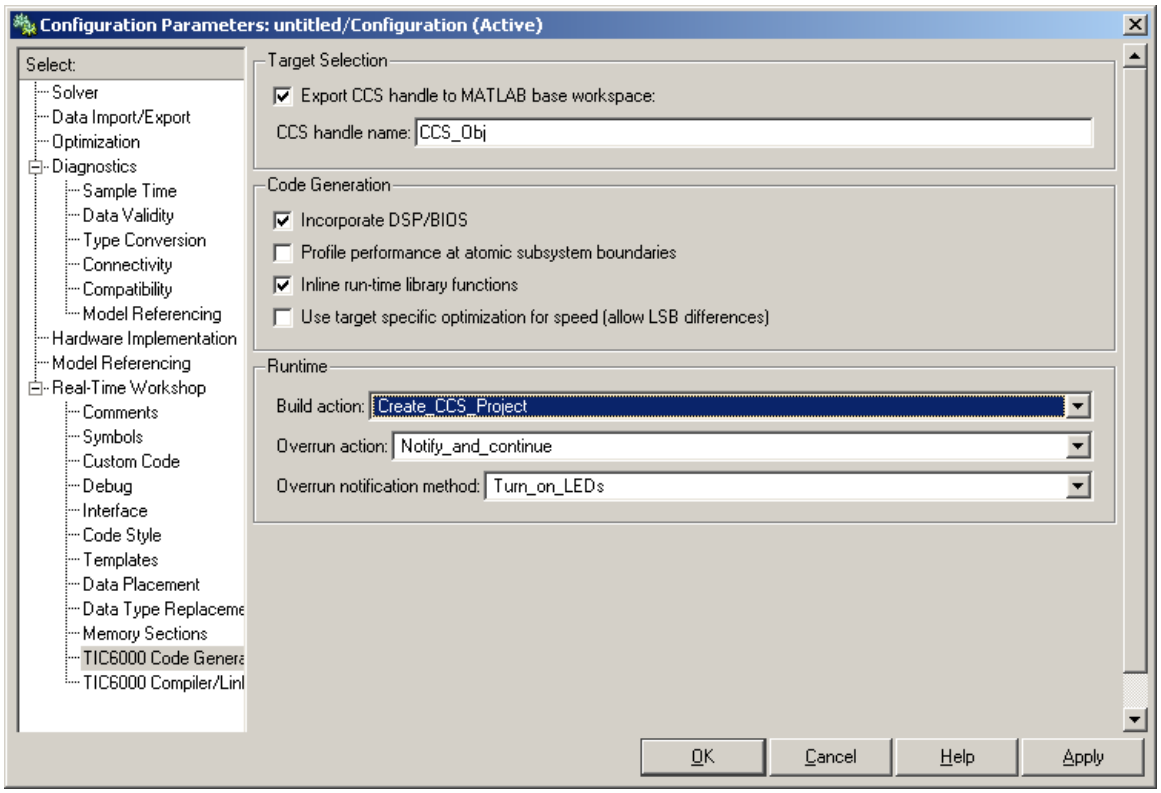


Figure 2: Configuration Parameters for CCS

We now need to create Simulink dedicated blocks to execute CCS commands. The first is the "Build". We can do this in many ways: clicking Build icon, pressing Ctrl-B or in MATLAB® typing make_rtw. Given this command is available in MATLAB, we will create a custom block with this command.

-
-
-
  - Give it a name. In our example we use "Generate Code & Create Project"
  - Choose the shadow, colour etc...
  - Right click on the Subsystem, choose " Block Properties". Go to the Callbacks tab, and under OpenFcn enter: "make_rtw;", as shown in the following picture:

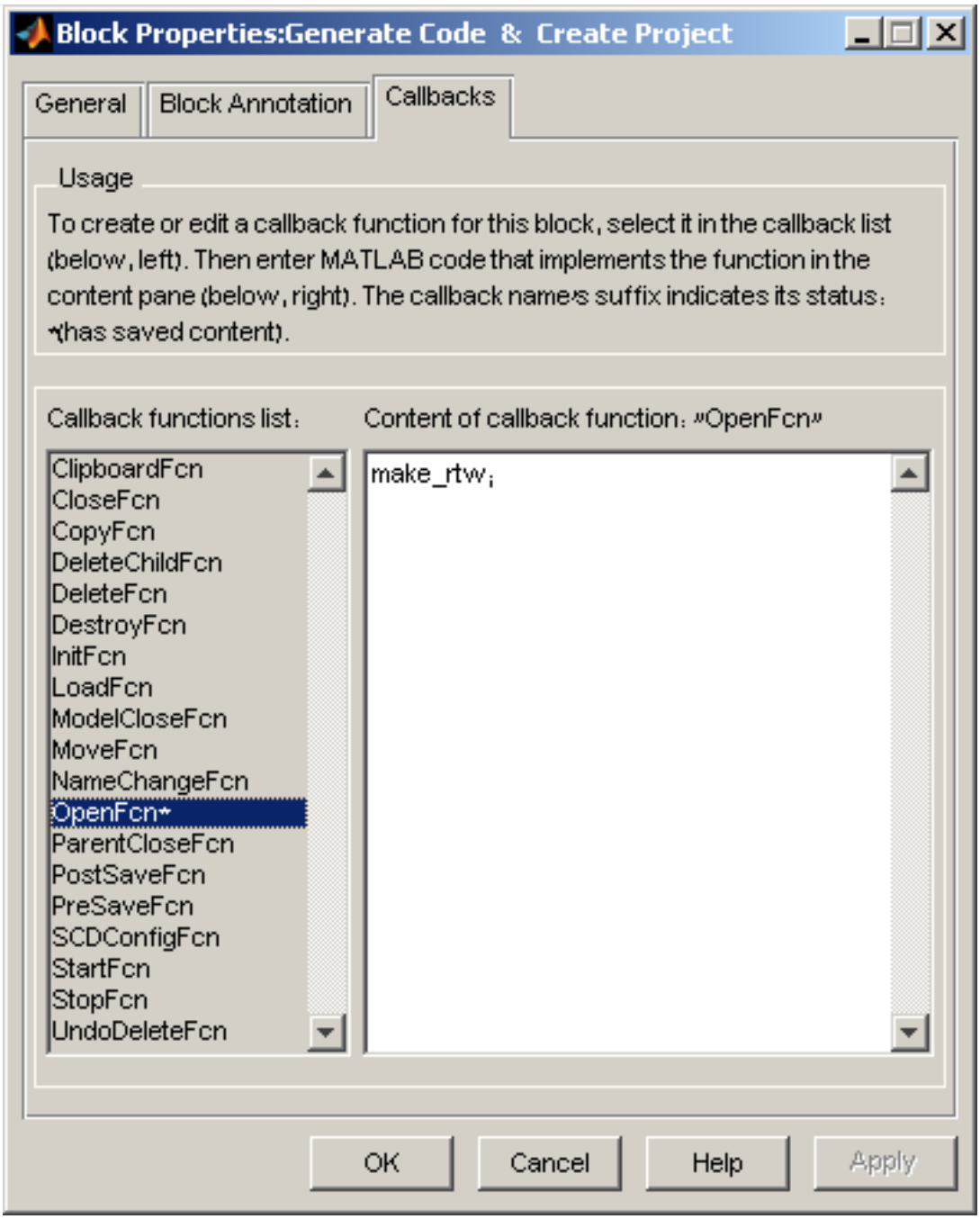


Figure 3: Creating the "Generate Code & Create Project" Simulink block

- Repeat this process for the following commands as follows:

Block Name	CCS Command	OpenFcn
Build	Build in CCS:	CCS_Obj.build;
Load	Load Program in CCS:	CCS_Obj.load([pwd,'\',bdroot,'_c6000_rtw\', bdroot, '.out']);
Run	Run on target:	CCS_Obj.run;
Halt	Halt on target:	CCS_Obj.halt;
Reset	Reset target:	CCS_Obj.reset;
Close Active Project	Close Active Project	close(CCS_Obj,[],'project')

Table 1

Your model should look as follows:

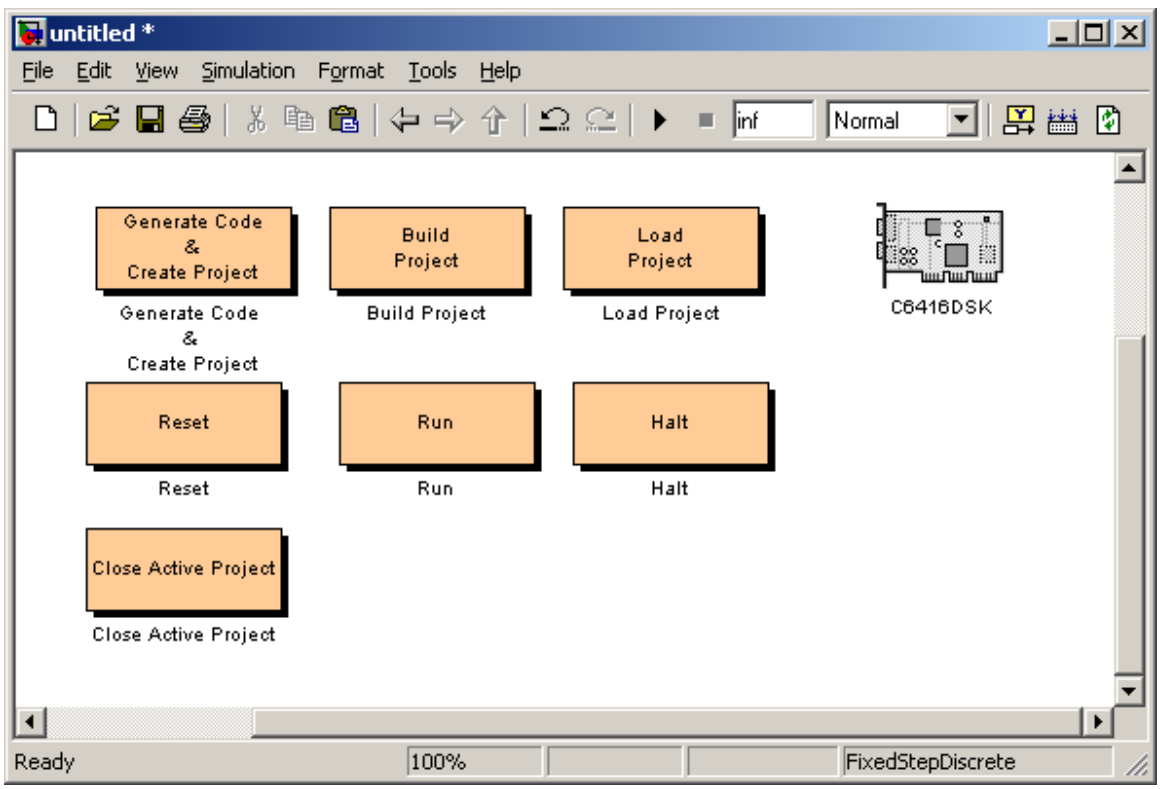


Figure 4: Simulink Blocks for CCS commands

3 Image (Stills) I/O

3.1 Simulation

Figure 5 shows the data flow for the edge detection simulation. The Edge detection model reads the source picture from a file in the PC, and displays both the original and the processed images.

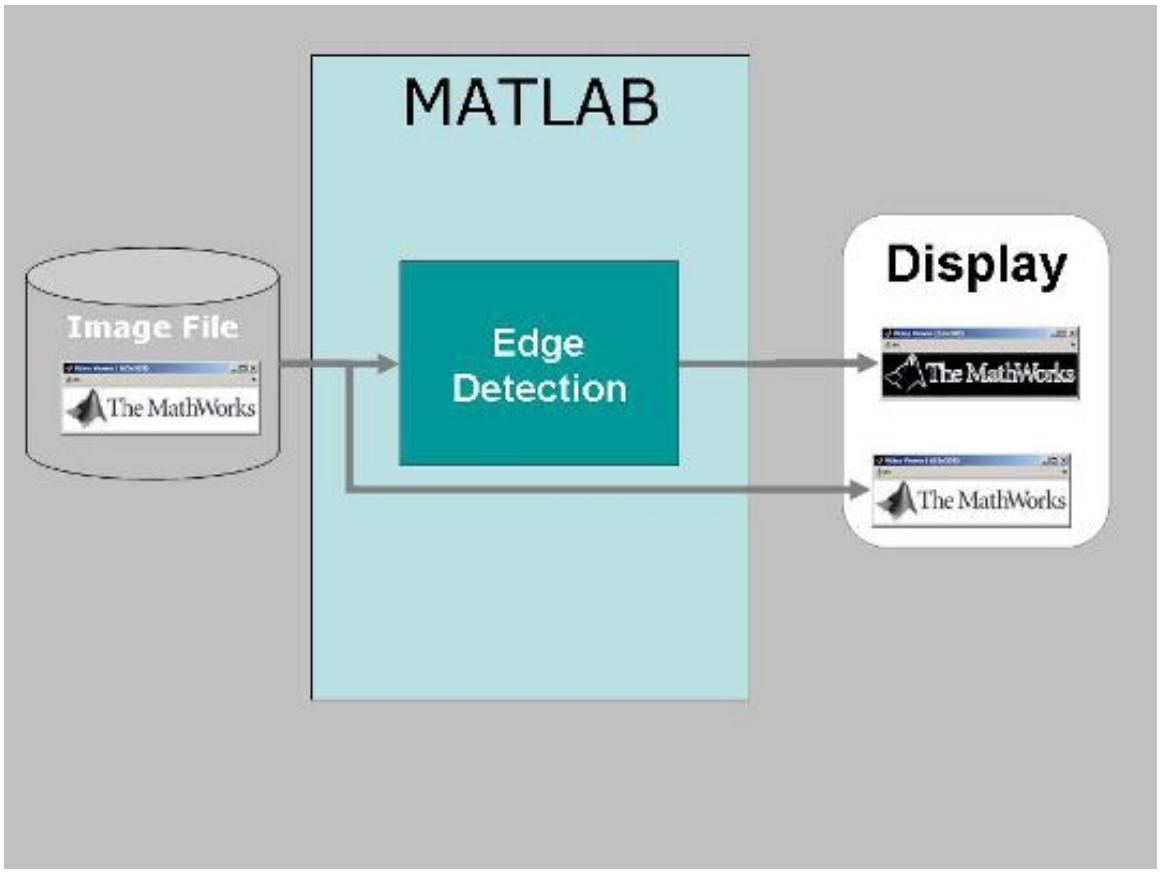


Figure 5: Simulation Environment

3.1.1 Reading and Displaying a Picture

1. Start by creating a new model in Simulink
2. Open the Simulink library browser and add the "Image from File" from the "Sources" group of the Video and Image Processing Blockset (Please refer to Figure 6). This block reads an image and converts it to the RGB format in Simulink .

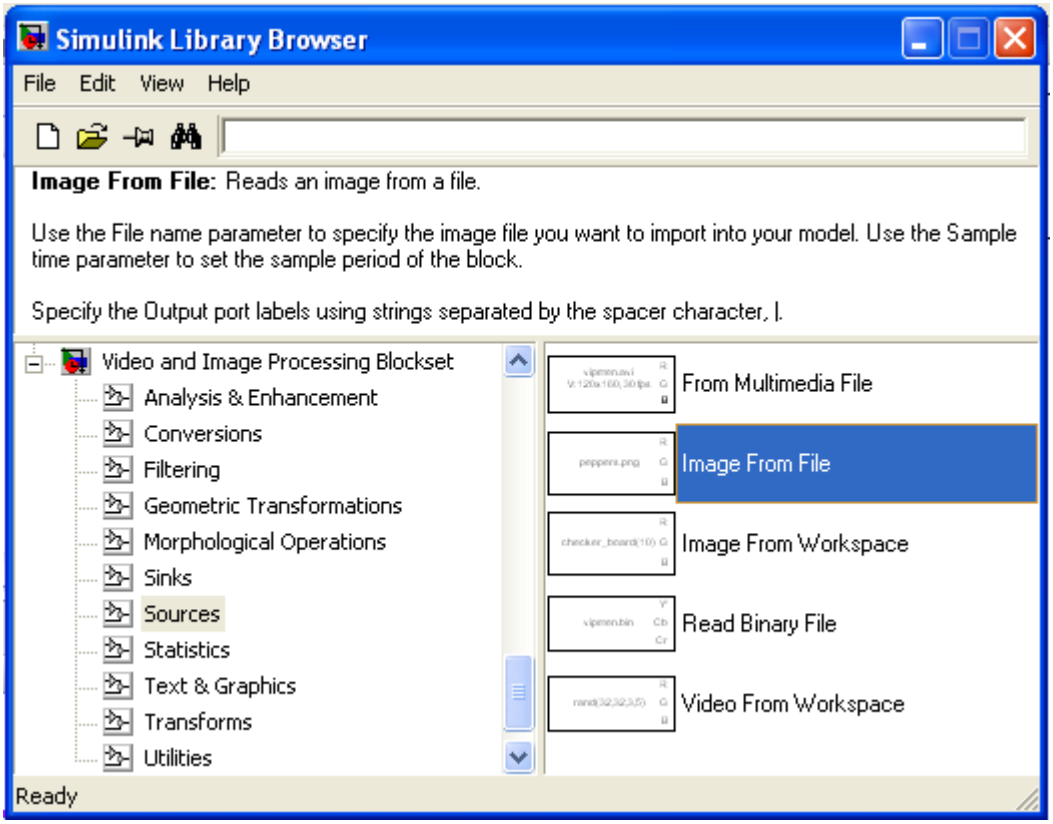


Figure 6: The "ImageFrom File" Simulink Block

3. Double Click and select the TMW2.jpg[5] image (Please refer to Figure 7).

[5]See the file at <http://cnx.org/content/m23996/latest/TMW2.jpg>

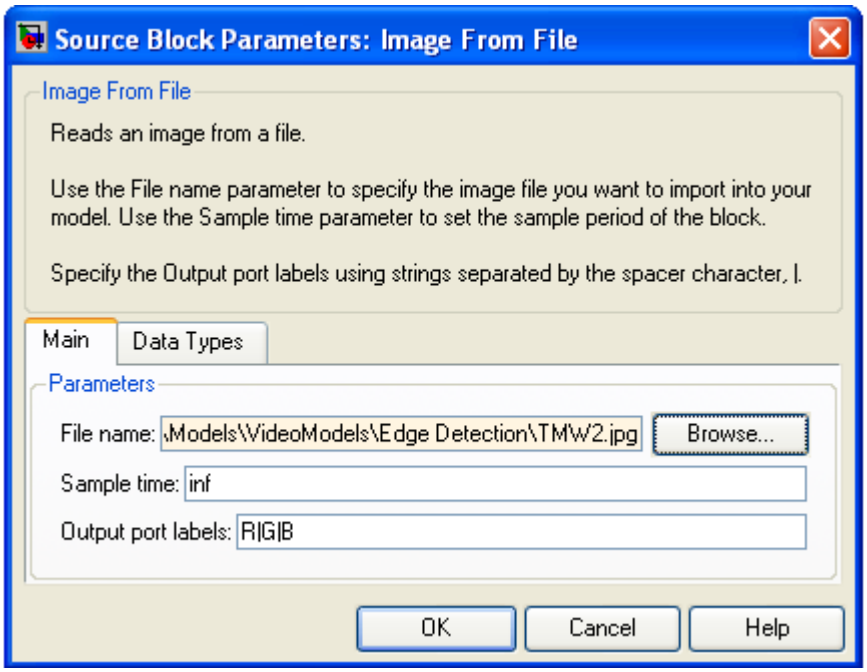


Figure 7: The Source Block Configuration

4. In order to display the picture the "Video Viewer" from the "Sinks" group of the Video and Image Processing Blockset (Please refer to Figure 8). This block displays an input RGB or intensity video stream or image. The edge detection block operates on intensity images.

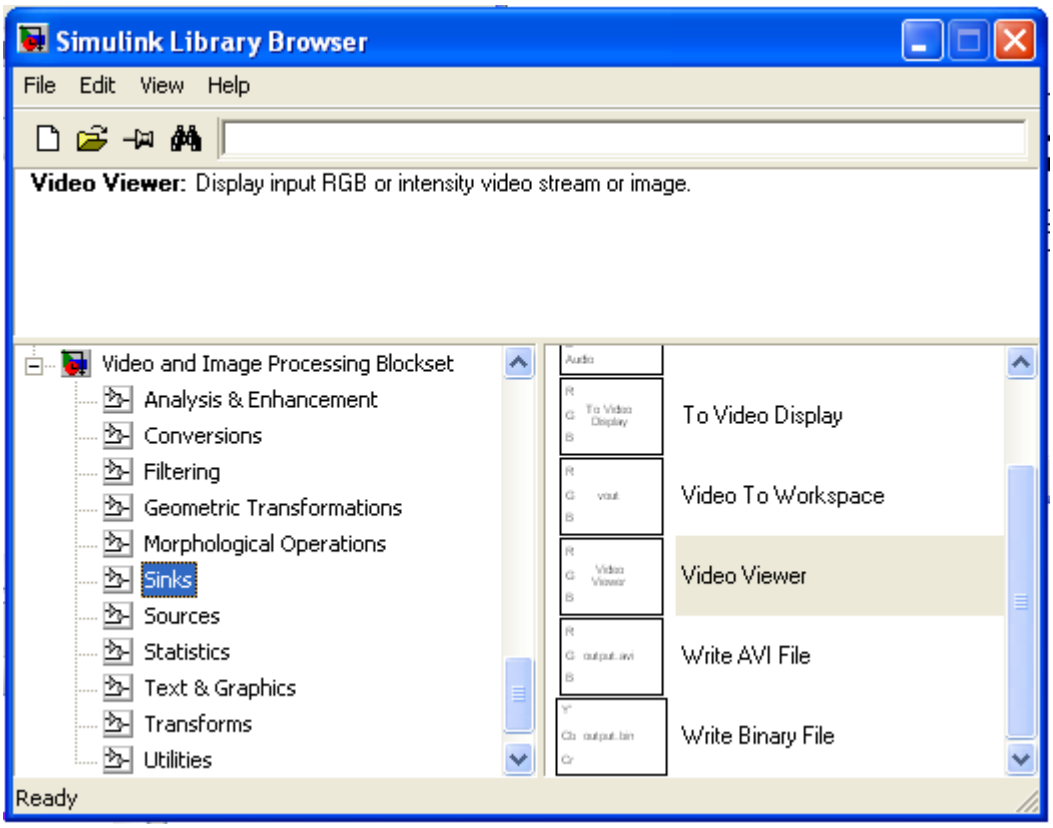


Figure 8: The Video Viewer Block

5. Select the "intensity" input format as shown in Figure 9.

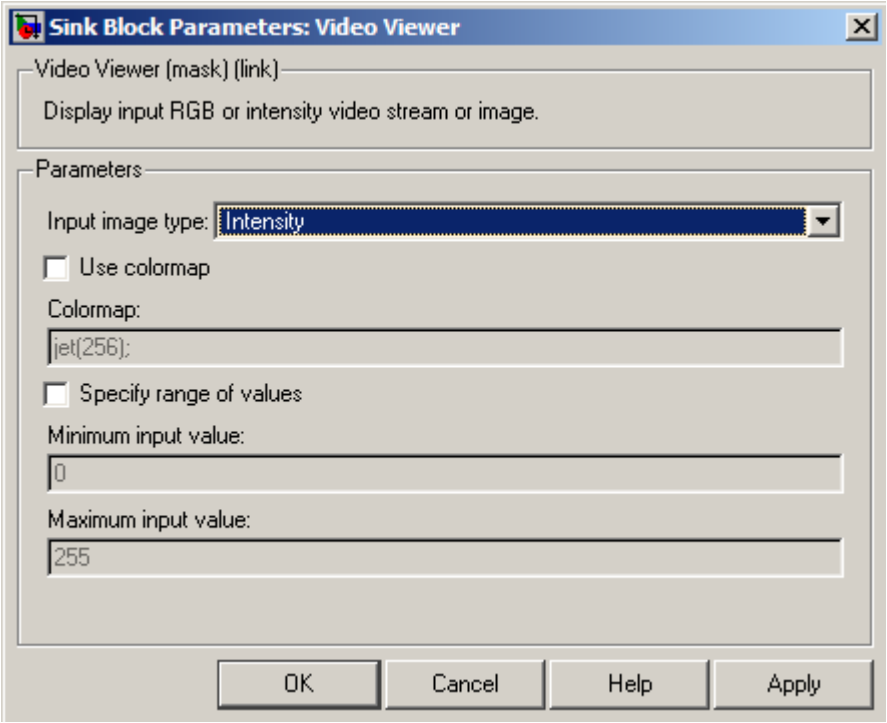


Figure 9: Video Viewer Configuration

6. The conversion from RGB to intensity will be executed in the Color Space Conversion: from the "Conversions" group of the Video and Image Processing Blockset (Please refer to Figure 10).

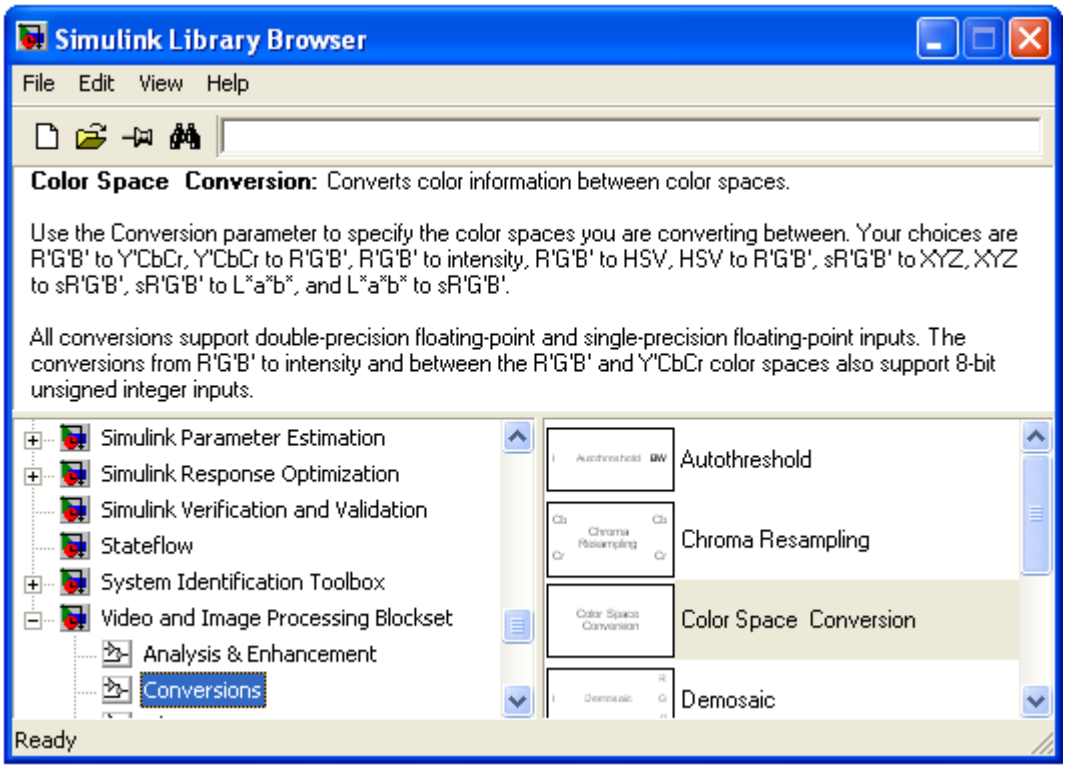


Figure 10: The Color Conversion Block

7. Double Click and select the "R'G'B' to intensity" conversion scheme as shown in Figure 11.

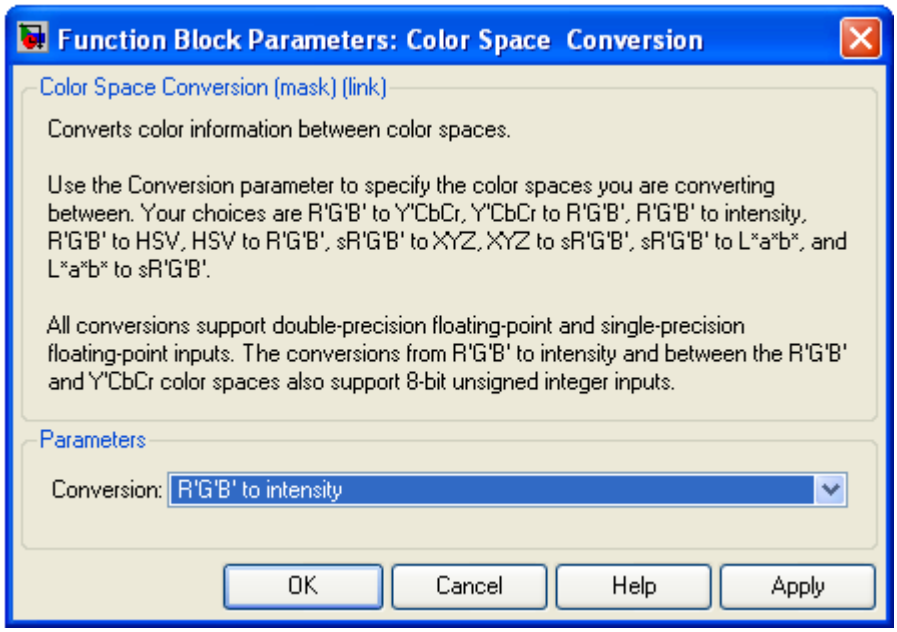


Figure 11: Configuring the Color Conversion Scheme

8. Connect the three blocks as shown in Figure 12.

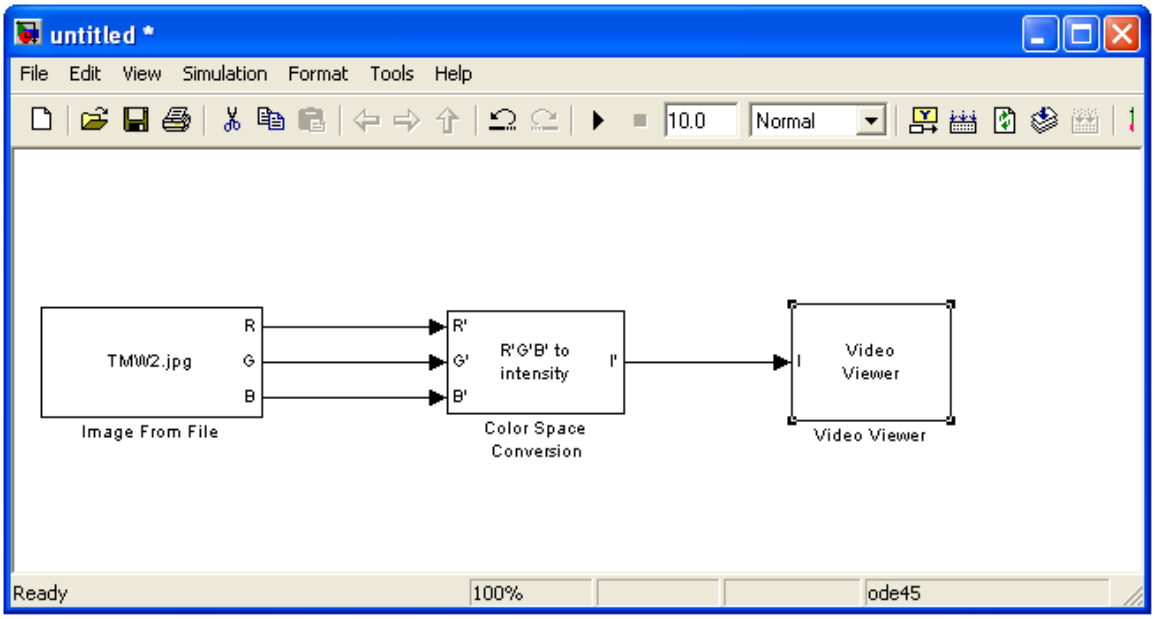


Figure 12: Reading and Displaying a Picture

9. Run the model, you should get the image shown in Figure 13

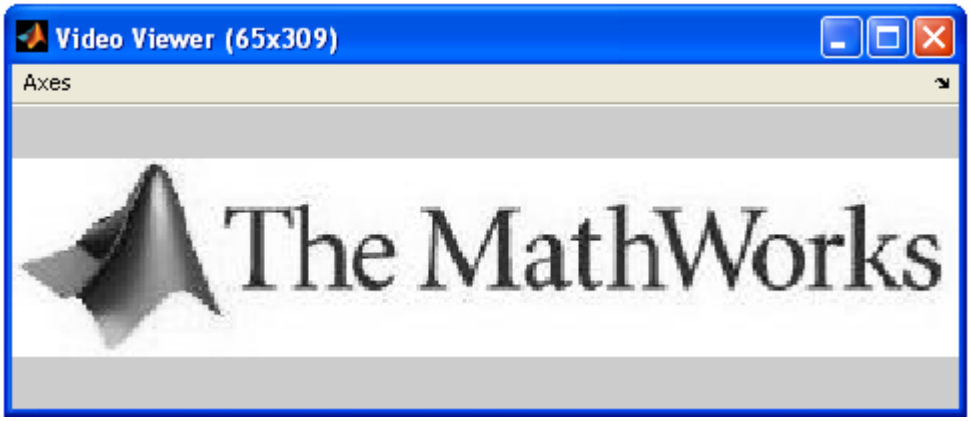


Figure 13: The Video Viewer Display

3.2 Real-Time

shows the data flow for the real time implementation of the image processing algorithms. The algorithms will be implemented using the DSK6416. A MATLAB will run simultaneously in the PC. This script will read the source picture from a file in the PC, convert it to grayscale and send it to the DSK6416 through an RTDX channel. The processed picture will be sent from the DSK6416 through a second RTDX channel. The script will display three images:

- The original color picture
- The grayscale picture
- The processed image

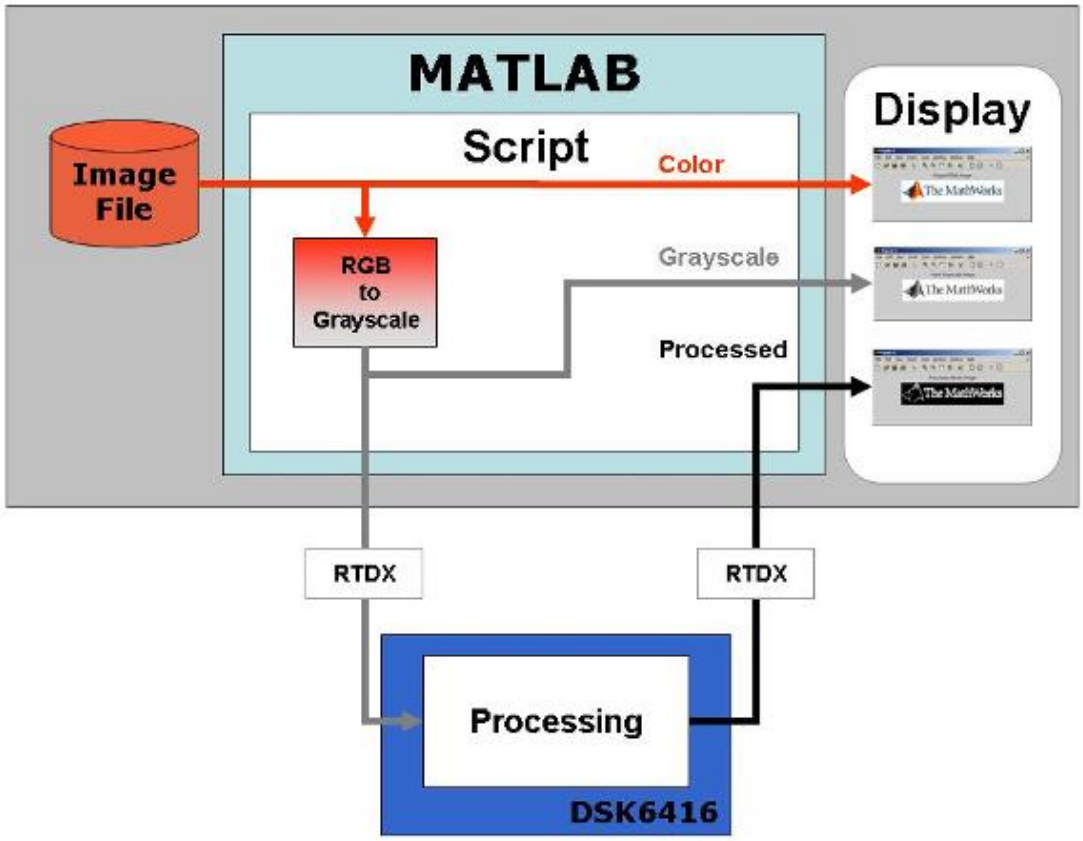


Figure 14: Real Time Implementation Environment

3.2.1 CCS Code Generation

Use the model created in chapter 2.

1. Add the "From RTDX" from the "RTDX Instrumentation" group of the "Embedded Target for the TI TMS320C6000 DSP" Blockset (Please refer to Figure 15). This block will be used receive data from the PC using a specified RTDX.

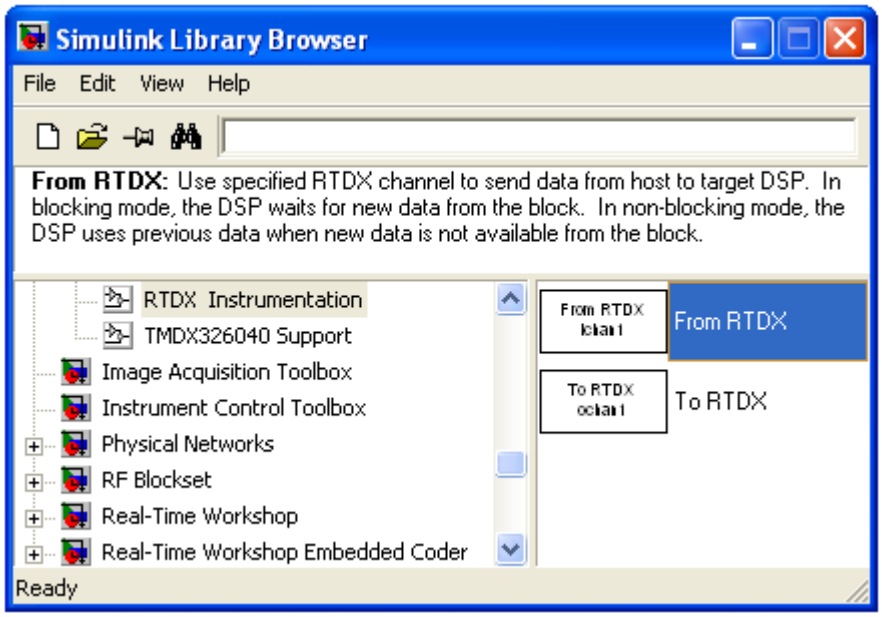


Figure 15: The "From RTDX" Block

2. Configure the block to receive data from a channel named "inputimage" as shown in Figure 16 [6]. For the test we will send in an image over RTDX[7]. The selection of 'Blocking Mode' will cause the code running on the DSP to wait until the RTDX transfer is complete before executing. This will give the MATLAB script full control over the execution.

[6] If you are using the XDS560 or the Blackhawk USB560, you will need to enable High Speed RTDX in the "C61416DSK" block.

[7] The "Output Dimensions" should contain the Frame size

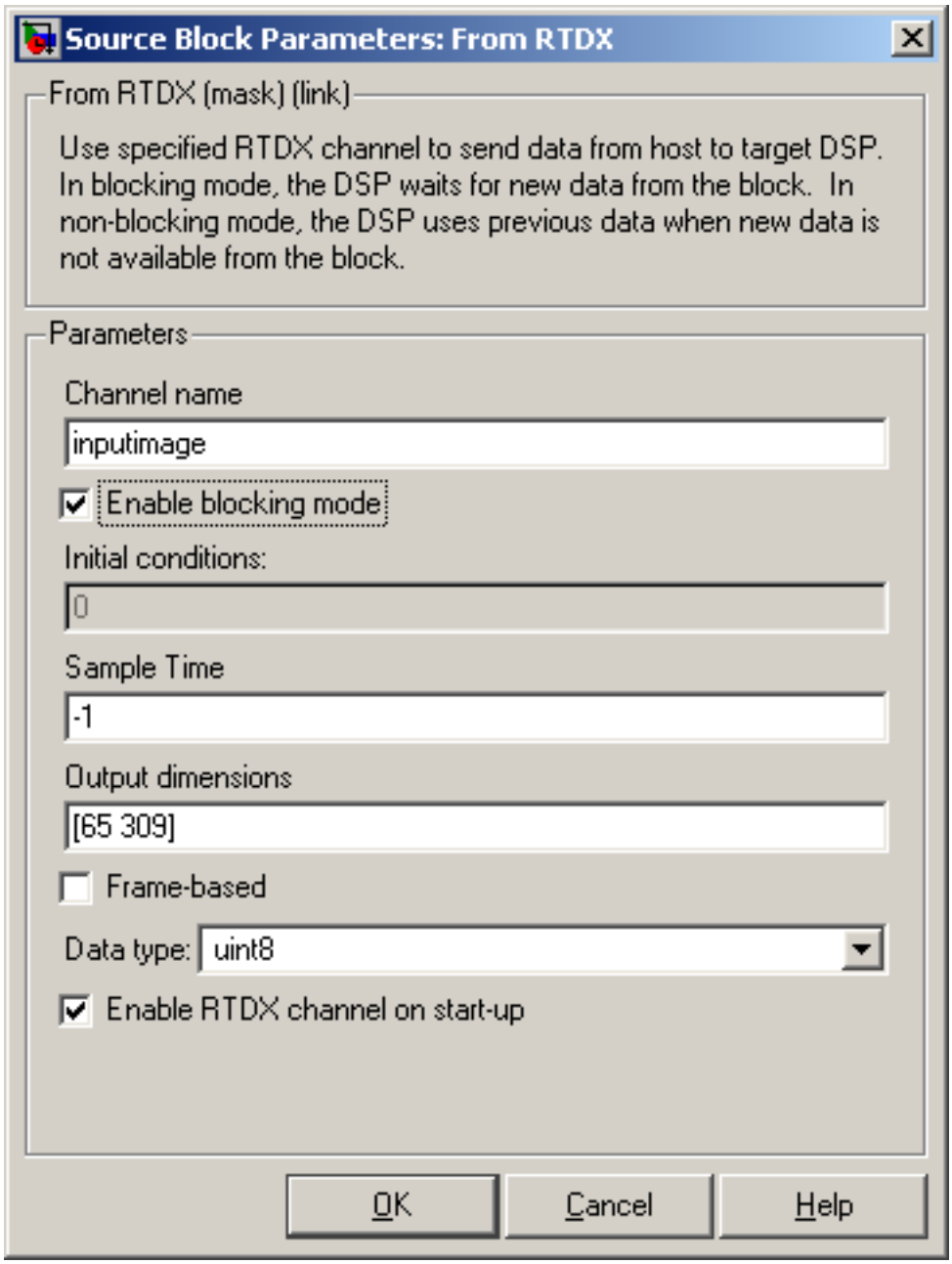


Figure 16: Configuring the Input Channel

3. Add the "To RTDX" from the "RTDX Instrumentation" group of the "Embedded Target for the TI TMS320C6000 DSP" Blockset (Please refer to Figure 15). This block will be used to send the processed image to the PC using a specified RTDX.

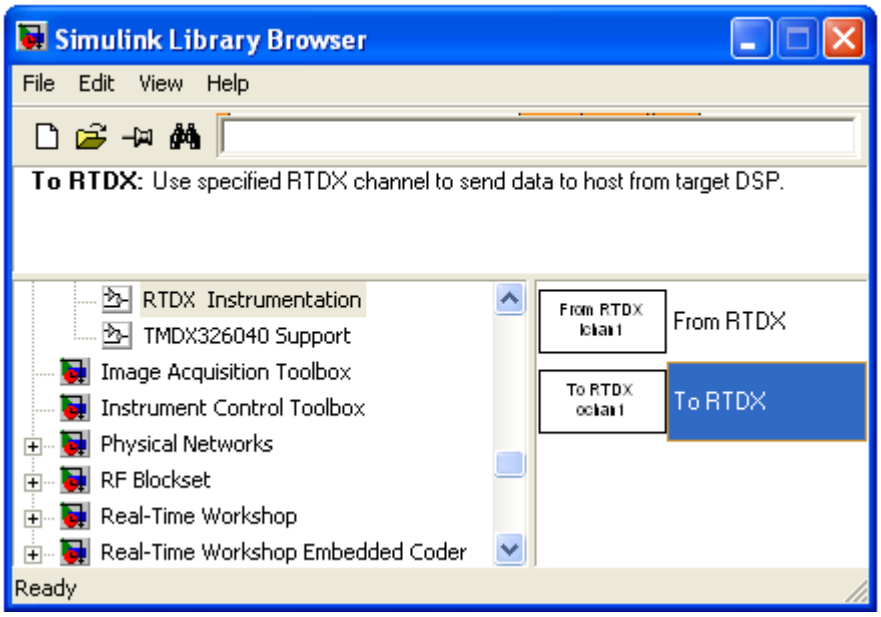


Figure 17: The "To RTDX" Block

Configure the block to send data through a channel named "outputimage" as shown in Figure 9.

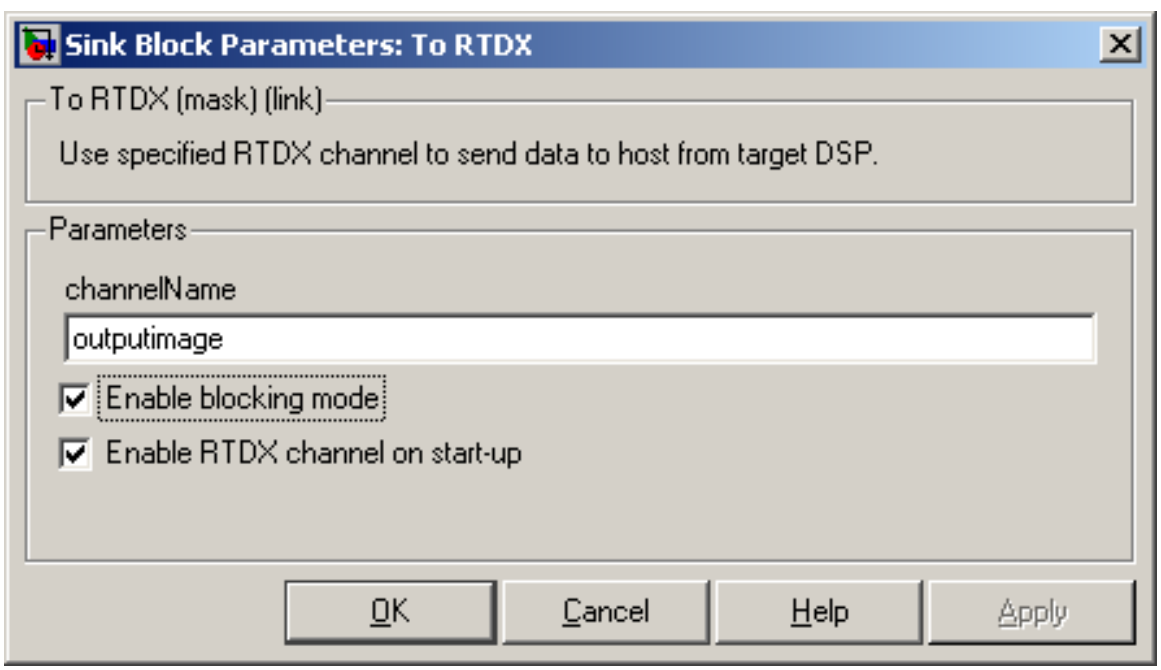


Figure 18: Configuring the Output Channel

4. The RTDX channel transfers data in the uint8 format. Therefore the output of the Edge Detection Block should be converted to this format. Add the "Convert Image" block from from the "Conversions" group of the Video and Image Processing Blockset, (Please refer to Figure 19). The block should be configured as shown in Figure 20.

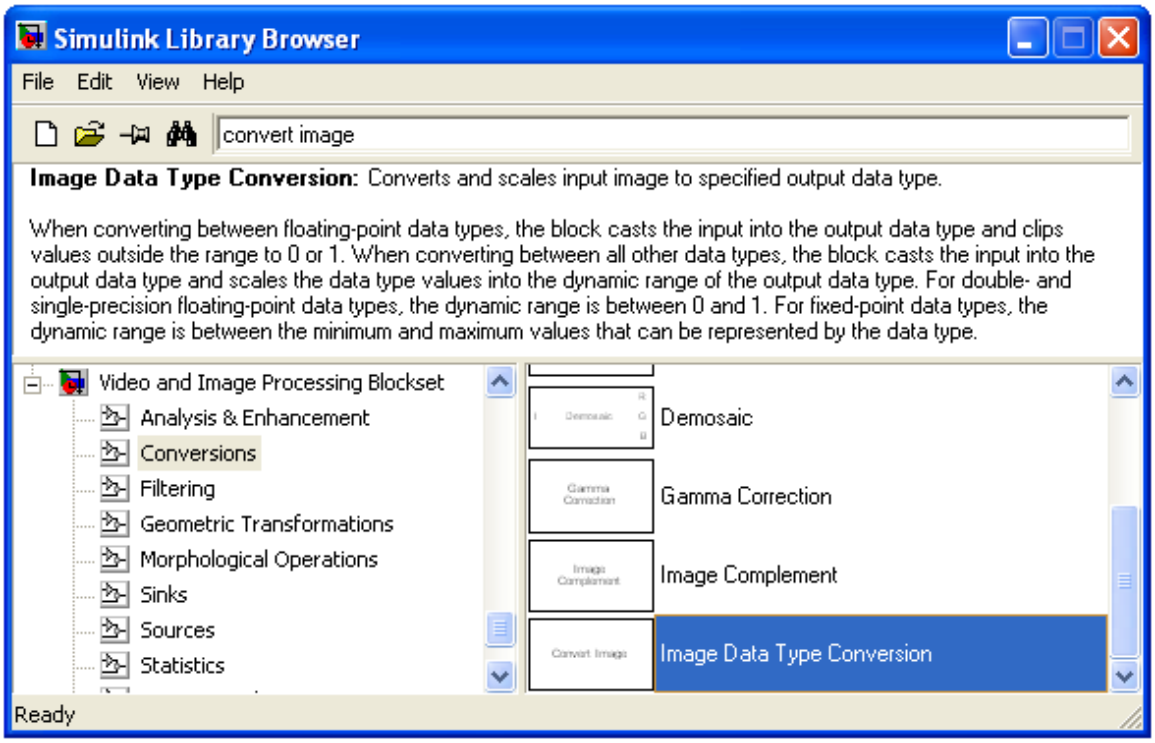


Figure 19: The Image Conversion Block

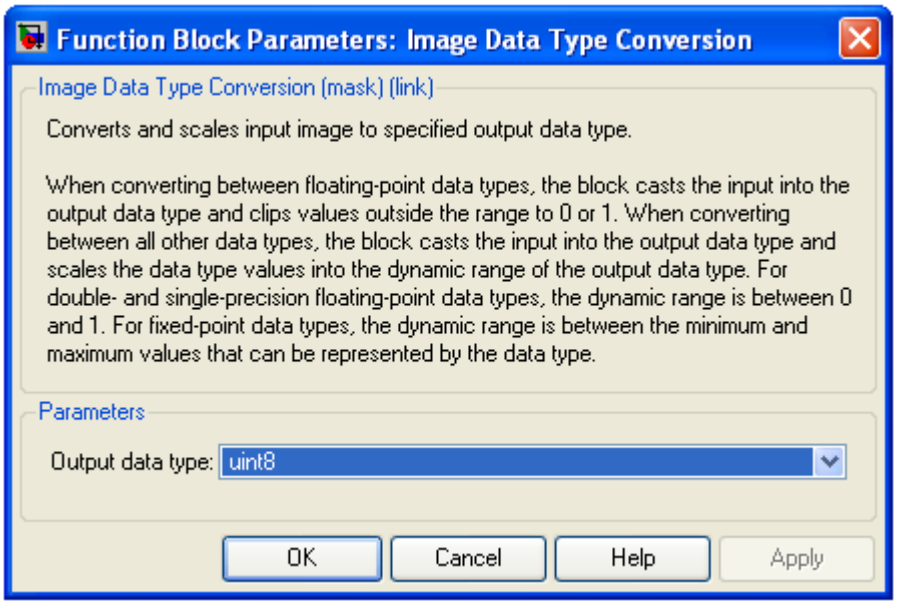


Figure 20: The Image Conversion Configuration

5. Save your model (We used the name "stills_R_W.mdl"). Your model should look as shown in Figure 21.

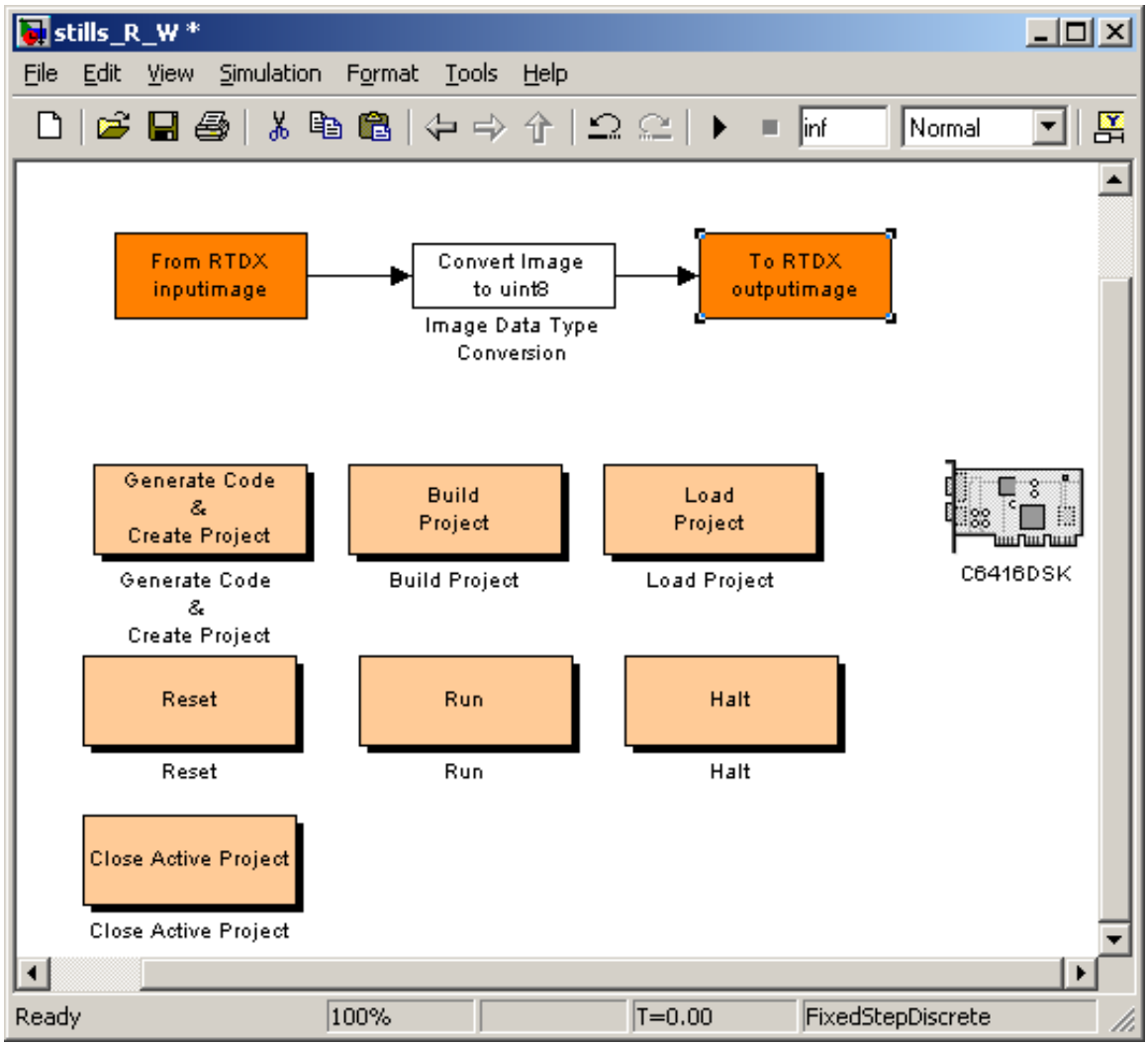


Figure 21: Reading and Displaying a Picture

6. Now you can generate the code to your DSK by using Ctrl-B[8]. When the model builds, it creates an object in the MATLAB workspace which represents CCS, by default called CCS_Obj. This object will be used to interact with the application on the DSP from MATLAB.
7. Build the project. Double-click the "Build Project" block.
8. Load the project. Double-click the "Load Project" block.
9. Run the target. Double-click the "Run" block. It should now be running, as displayed in CCS:

[8] If you do so, the build will fail at compilation in CCS due to an ISRAM space issue. In this case, we can fix this by redefining the location for .far as SDRAM rather than ISRAM (the default) in the board properties box. Rebuild, and it should compile fine.

This media object is a downloadable file. Please view or download it at
<http://cnx.org/content/m23996/1.1/fig22.JPG>

Figure 22: CCS Compilation Display

3.2.2 The MATLAB Script

The purpose of the MATLAB code we want is to define an interface with the RTDX channels, to control the execution of the application on the processor, to send/retrieve data from the processor and to display it. An example script, "picture_script.m" will do this. The steps are also detailed below:

1. With the application running, we can configure the RTDX channels in CCS.

```
%% Configure RTDX channels
% Identify RTDX channel names/modes
chan_struct(1).name = 'inputimage';
chan_struct(1).mode = 'w';
chan_struct(2).name = 'outputimage';
chan_struct(2).mode = 'r';
% Identify RTDX host buffer parameters
RTDX_config_struct.Buffsize = 32768;
RTDX_config_struct.Nbuffers = 4;
RTDX_config_struct.Mode = 'continuous';
% Set up RTDX
r = setupRTDX(CCS_Obj, chan_struct, RTDX_config_struct);
```

   This This sets up a writeable RTDX channel to inputimage and a readable RTDX channel to outputimage. The names used are those in the blocks in the Simulink model.
2. Once running, we enable the RTDX channels for use, and get some data to send to the DSP – in this case a grayscale image taken from a file.

```
%% Enable RTDX
r.enable('all');
%% Import an image from file to send to the DSP
Y = imread('TMW2.jpg');
I = rgb2gray(Y);
```

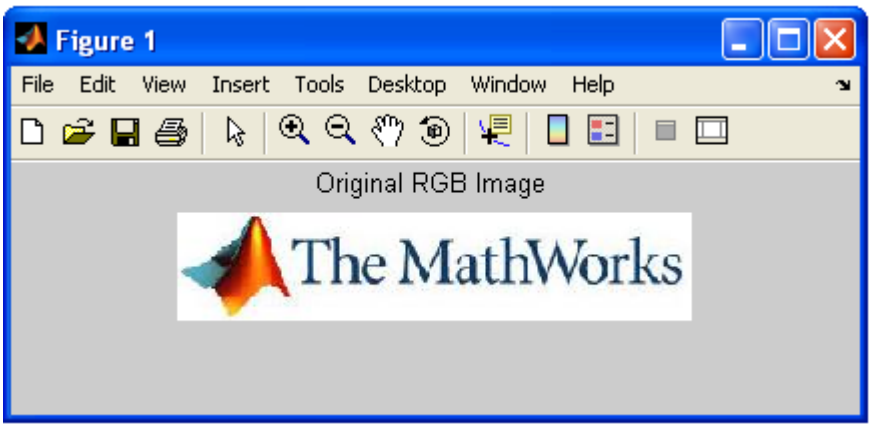
3. We then send this to the DSP:

```
%% Send test frame to target via RTDX
r.writemsg('inputimage', I);
```

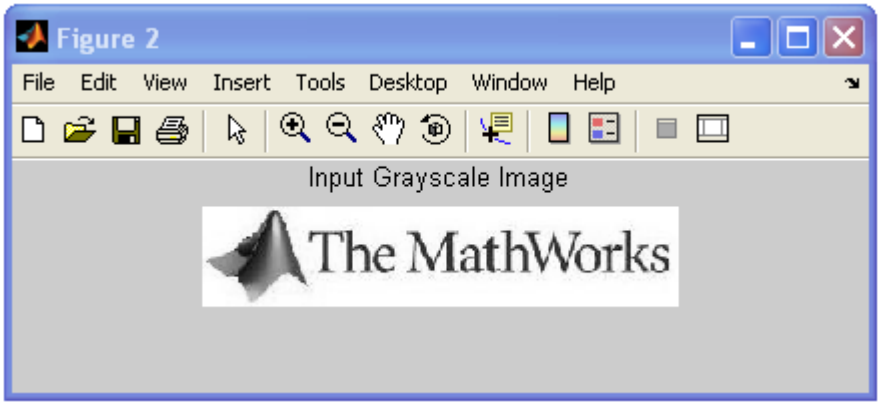
   On receipt of this message, the DSP will process the data and send it back over the return RTDX channel, and then wait for further input.
4. We can read the processed data back:

```
    %% Read processed test frame from DSP
O = r.readmsg('outputimage', 'uint8', size(I));
```

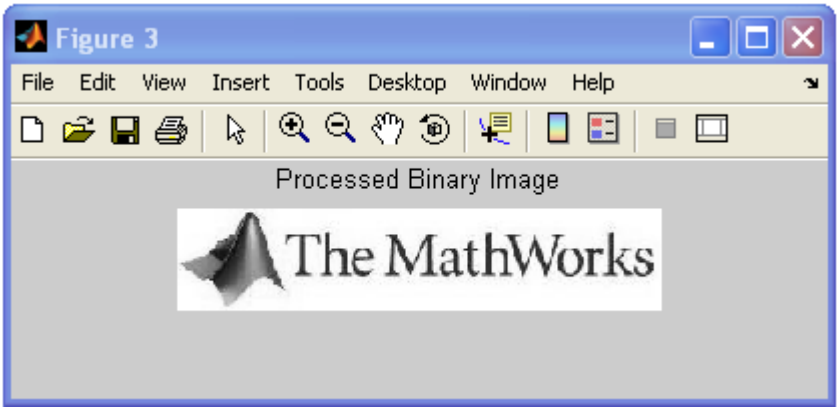
5. This script is saved under "Picture_script.m"
6. Run the script and you will see:



(a) The Original Color Picture



(b) The Correspondent Grayscale Picture



(c) The Received Image (Loopback)

Figure 23: Image Processing Display